METHOD FOR MAPPING FACIAL ANIMATION VALUES TO HEAD MESH POSITIONS

BACKGROUND OF THE INVENTION

The present invention relates to head animation, and more particularly, to generating an animated three-dimensional video head based on two-dimensional video images.

Virtual spaces filled with avatars are an attractive way to allow for the experience of a shared environment. However, animation of a photo-realistic avatar generally requires intensive graphic processes, particularly for rendering facial features.

Accordingly, there exists a significant need for improved rendering of facial features. The present invention satisfies this need.

SUMMARY OF THE INVENTION

The present invention provides a technique for translating an animation vector to a target mix vector. In the method, a calibration vector is generated and the animation vector is mapped to the target mix vector using the calibration vector.

Other features and advantages of the present invention should be apparent from the following description of the preferred embodiments taken in conjunction with the accompanying drawings, which illustrate, by way of example, the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic flow diagram showing a technique for translating an animation vector to a target mix vector, according with the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides a technique for translating an animation vector to a target mix vector.

With reference to FIG. 1, the animation of an avatar is defined by a set of targets. A mapping algorithm provides the translation from animation vector 12 to target mix vector.

The animation vector is the abstracted sensing result. It is the most compact representation of the facial expression as determined by audio-visual sensing.

By definition, the animation vector is zero for the neutral expression.

The target mix vector describes the contribution of each individual target to the current expression.

Different mapping algorithms may be used. Their common goal is to provide a reasonable interpolation between the points in animation space associated with the targets. Each mapping algorithm is exactly defined by a set of parameters. The parameters vary from algorithm to algorithm.

Calibration may be performed by multiplying the target mix vector with a diagonal matrix. Since the matrix is diagonal, it is henceforth referred to as the calibration vector.

An overview of the translation from animation vector to target mix vector follows. Be $\underline{a}$ the animation vector of dimension N_a (number of animation values), $\underline{g}$ the target mix vector of dimension M (number of independent targets) and $p_1,...,p_L$ the parameters of mapping algorithm $\underline{F}()$, then

$$\underline{g} = \underline{F}(\underline{a}, p_1,...,p_L) \quad (1)$$

The calibrated target mix vector is obtained by multiplying with the diagonal matrix defined by the calibration vector $\underline{c}$:

$$\underline{g}_c = \begin{pmatrix} c_1 & & & \\ & c_2 & & \\ & & \ldots & \\ & & & c_M \end{pmatrix} \cdot \underline{g} \quad (2)$$

Further, be $\underline{t}_1,....,\underline{t}_M$ the parameterization of the targets associated with the components of the target mix vector and $\underline{t}_0$ the parameterization of the neutral model, then the parameterization of the current expression can be obtained by a simple matrix multiplication:

(3) $$\underline{t} = \left(\underline{t}_1 - \underline{t}_0 \quad \underline{t}_2 - \underline{t}_0 \quad \ldots \quad \underline{t}_M - \underline{t}_0\right) \cdot \underline{g}_c + \underline{t}_0$$

The matrix $\left(\underline{t}_1 - \underline{t}_0 \quad \underline{t}_2 - \underline{t}_0 \quad \ldots \quad \underline{t}_M - \underline{t}_0\right)$ is referred to as the *target matrix* $\underline{\underline{T}}$.

A description of the mapping algorithm follows. Every target $\underline{t}_i$ is associated with an animation vector $\underline{a}_i$. The target and animation vectors are connected by this deliberate association only. This will become obvious in the formulas, where targets enter only as the *i*-th unity vector $\underline{e}_i$ representing the target mix vector that results in exactly that target. The parameterization of the target $\underline{t}_i$ is not relevant for computation of the mapping parameters. (This means that is does not matter if the target is defined by vertex positions, morph link positions or muscle tensions.)

The animation vector can be set manually or it can be derived from a reference model with targets, if the model is equipped with ground truth anchors that enable the application of the tracking algorithm to the model and it's deformations AND if the reference model implements all needed targets. The reference model must have a human geometry, since the purpose of the ground truth anchors is to simulate tracking on the model. Manual editing is necessary if the animation vector contains elements that cannot be derived from visual sensing, such as lip synch animation values.

The mapping is basically a multidimensional interpolation in $\underline{a}$ between the target points. The mapping parameters are determined by minimizing the error in reproducing the target

points. Depending on the mapping algorithm, perfect reproduction of the target points may not be possible.

The parameters $p_1,...,p_L$ of the mapping are determined by solving the set of equations

$$(4) \qquad \frac{\partial}{\partial p_j}\sum_{i=1}^{M}\left\|\underline{e}_i - \underline{F}(\underline{a}_i, p_1,..., p_L)\right\|^2 = 0, \quad \forall j \in [1, L]$$

Targets can be divided into independent groups of targets, such as eye region targets and mouth region targets. Different mapping algorithms can be applied to the different groups to achieve more flexibility.

A description of types of mapping algorithms follow. The simplest mapping algorithm is the linear mapping:

$$(5) \qquad \underline{F}(\underline{a}, \underline{\underline{P}}) = \underline{\underline{P}} \cdot \underline{a}$$

The parameter matrix is determined by solving the equation

$$(6) \qquad \underline{\underline{P}} \cdot (\underline{a}_1 \quad \underline{a}_2 \quad ... \quad \underline{a}_M) = \underline{\underline{1}}$$

using singular value decomposition. If $N_a < M$, the equation (6) is overdetermined and SVD will return the "solution" that satisfies eq. (4). If $N_a > M$, the equation is underdetermined

and the SVD solution will be the vector with the smallest norm that satisfies equation (6). SVD is described in great detail in "Numerical Recipes".

The linear method is internally referred to as the "matrix method" because of the form of the algorithm parameters.

A more general mapping is achieved by using a set of basis functions as input. Obviously, the linear method is a special case of this more general method.

$$\underline{F}(\underline{a}, \underline{\underline{P}}) = \underline{\underline{P}} \cdot \underline{B}(\underline{a}) \tag{7}$$

The solution is analog to the solution of the linear problem. Since the number of basis functions is independent of the number of animation values N_a, it is always possible to choose exactly M functions, so that the system is neither over- or underdetermined:

$$\underline{\underline{P}} = \left(\underline{B}(\underline{a}_1) \quad \underline{B}(\underline{a}_2) \quad \ldots \quad \underline{B}(\underline{a}_M)\right)^{-1} \tag{8}$$

The basis functions can be chosen manually by carefully analyzing the animation vectors of the participating targets. This is the currently deployed method. The internal name is "matrix2" because it is an extension of the linear method.

The following set of basis functions and targets may be used. The basis functions are commonly referred to as "animation tag expressions".

Eye/eyebrow group basis functions (formulated in UPN)

```
eyeOpen,eyeAsym,-,|-
eyeOpen,eyeAsym,+,|-
eyeOpen,|+
eyeBrowRaise,eyeBrowAsym,-,|+
eyeBrowRaise,eyeBrowAsym,+,|+
eyeBrowRaise,|-
```

Eye/eyebrow group targets:

```
MTEyeCloseR
MTEyeCloseL
MTEyeOpenWide
MTBrowRaiseR
MTBrowRaiseL
MTBrowFurrow
```

Mouth group basis functions

```
lipDistance,mouthVertPos,-
lipDistance,mouthVertPos,+,|+
lipDistance,mouthVertPos,+,|-
mouthWidth,|-
```

```
mouthWidth,|+
mouthCornerUp,|+
mouthCornerUp,|-
mouthHorizPos,0.5,mouthRot,*,-,|+
mouthHorizPos,0.5,mouthRot,*,-,|-
visemeB
visemeF
visemeS
```

Mouth group targets:

```
MTMouthAh
MTMouthDisgust
    MTMouthDown
    MTMouthOh
    MTMouthEe
    MTMouthSmileOpen
    MTMouthFrown
    MTMouthPullL
    MTMouthPullR
    MTMouthB
    MTMouthF
```

MTMouthS

Each basis function is designed to best match to a specific target. The order in the table above represents that match. It is very tedious to design a basis function manually such that it only responds when the associated target is acted and not responds when any other target is acted. Off-diagonal elements of the *P* matrix lead to corrections and decouple the targets such that this desired behavior is achieved.

The target matrix, calibration matrix and mapping parameter matrix can be combined into one matrix by simple multiplication, which can be done ahead of time:

$$(9) \qquad \underline{t} = \underline{\underline{T}} \cdot \underline{g}_c + \underline{t}_0 = \underline{\underline{T}} \cdot \begin{pmatrix} c_1 & & & \\ & c_2 & & \\ & & \cdots & \\ & & & c_M \end{pmatrix} \cdot \underline{\underline{P}} \cdot \underline{B}(\underline{a}) = \underline{\underline{D}} \cdot \underline{B}(\underline{a}) + \underline{t}_0$$

$$\underline{\underline{D}} = \underline{\underline{T}} \cdot \begin{pmatrix} c_1 & & & \\ & c_2 & & \\ & & \cdots & \\ & & & c_M \end{pmatrix} \cdot \underline{\underline{P}}$$

The decoder matrix $\underline{\underline{D}}$, offset vector (neutral target) $\underline{t}_0$ and definition of basis functions $\underline{B}(\underline{a})$ are the parameters of the animation decoder used in the various players.

A description of radial/angular basis function mapping follows. The basis functions can also be a set of automatically determined functions such as radial basis functions. Certain properties of the mapping algorithm can be designed into the basis functions. A desirable property of the algorithm is scale linearity:

(10) $\underline{F}(\lambda \underline{a}) = \lambda \underline{F}(\underline{a})$

This means that the "amplitude" of an expression is translated linearly to the model. It can be obtained by using basis functions of the form

(11) $$b_i(\underline{a}) = \frac{\|\underline{a}\|}{\|\underline{a}_i\|} \tilde{b}_i\left(\left\| \frac{\underline{a}}{\|\underline{a}\|} - \frac{\underline{a}_i}{\|\underline{a}_i\|} \right\| \right)$$

Because of their linear or locally linear behavior, the following functions are useful for $\tilde{b}(\vartheta)$:

(12) $\tilde{b}(\vartheta) = \vartheta$

(13) $\tilde{b}(\vartheta) = \frac{1}{\alpha} \arctan(\alpha \vartheta)$

(14) $\tilde{b}(\vartheta) = \frac{\vartheta}{1 + \alpha \vartheta}$

(13) and (14) can be locally approximated by (12) and have interesting saturation characteristics. The parameter α determines the localization of these basis functions.

All mapping algorithms are somewhat arbitrary because of their nature as interpolation algorithms. The way interpolation is done between targets is determined by the choice of basis

functions. It seems to be reasonable to shoot for the algorithm that delivers the most linear interpolation possible while still reproducing all targets.

If the basis functions are determined automatically, it is easily possible to add K dependent targets that are created by linear superposition of independent targets. This enables one to have more control over the interpolation process by providing additional support points. Eq. (4) is then generalized to:

$$(15) \quad \frac{\partial}{\partial p_j} \sum_{i=1}^{M+K} \left\| \underline{g}_i - \underline{F}(\underline{a}_i, p_1, \ldots, p_L) \right\|^2 = 0 \quad \forall j \in [1, L]; \quad \underline{g}_i = \underline{e}_i \; \forall i \le M$$

Each dependent target is defined by its animation vector $\underline{a}_i$ and target mix vector $\underline{g}_i$ with $i > M$, which defines the superposition of independent targets. Eq. (8) has to be modified to yield the solution to this more general problem:

$$(16) \quad \underline{\underline{P}} = \left(\underline{g}_1 \quad \underline{g}_2 \quad \ldots \quad \underline{g}_{M+K}\right) \cdot \left(\underline{B}(\underline{a}_1) \quad \underline{B}(\underline{a}_2) \quad \ldots \quad \underline{B}(\underline{a}_{M+K})\right)^{-1}$$

A description follows of code to implement the animation technique outlined before. The implementation consists of authoring and player parts. The description covers the authoring part only up to and including the container for player decoder parameters. The code can be separated into classes and functions that are directly used by applications and classes that are only used internally. Throughout the descriptions, indices to either targets or animation values are replaced by string annotations (names). This allows for a more flexible configuration. Example: Instead of referring to target 3 or animation value 5, names like "MTMouthAh" or "lipDistance" are used in the interface. The actual indices may vary from object to object. Internally the names are used to build index lookup tables to perform the various linear algebra operations efficiently.

A description of the interface follows:

`aut_KeyframeData`

A pure container class to store target parameterizations ($\underline{t}_i$) and ground truth data for the purpose of computing animation vectors. Contains additional annotation information, such as the meaning of certain model parameters and names of animation values.

This class serves as input for the computation of the mapping parameters and for the computation of the decoder matrix.

`aut_LinTargetMapper`

A facade for the computation of a linear or manual basis function mapping. Contains necessary configuration parameters to define basis functions and participating targets.

Input is an `aut_KeyframeData` object (ground truth data only), output is an `aut_MappingResult` object.

`aut_MappingResult`

A container class to store the mapping parameters comprised of the mapping parameter matrix $\underline{\underline{P}}$, the definition of the basis functions $\underline{B}(\underline{a})$ (animation tag expressions) and the

calibration vector $\underline{c}$. Contains code to compute the calibrated target mix vector $\underline{g}_c$. Additional annotation information is contained to associate target names to target mix vector components.

`aut_computeAnimTagTrafo`

A function to perform the multiplication of the target matrix, calibration vector and mapping parameter matrix to obtain the decoder matrix. Input to the function is an `aut_KeyframeData` object (to obtain the target matrix, does not need ground truth data) and an `aut_MappingResult` object. The output is an `att_AnimTagTrafo` object as described below.

`att_AnimTagTrafo`

A container class to hold the decoder parameters, comprised of decoder matrix, offset vector and definition of basis functions (animation tag expressions). Functions provided are: computation of the model parameterization given an arbitrary animation vector and transform of the model parameterization into a different coordinate system.

A description of internal components follows:

`aut_LabeledFltMatIf` and derived classes

`aut_LabeledFltMatIf` is a pure interface to a "labeled float matrix". Each row and column of the matrix is annotated with strings to identify the data stored in the matrix.

Read/write access can be done conveniently by providing names instead of indices. An actual implementation of the container is `aut_LabeledFltMat`, which is used to store intermediate data such as animation vectors.

`aut_ExprExpLFMIf` is a proxy to a `aut_LabeledFltMatIf` and provides the functionality to compute mathematical expressions of matrix elements. This is used to compute basis functions/animation tag expressions. Instead of supplying the name of a matrix component, an expressions is passed.

`aut_MapBuilder` and derived classes

provide functionality to compute mapping algorithm parameters. The derived class `aut_LinearMapBuilder` is used in `aut_LinTargetMapper`. The class is configured on the fly from the parameters of `aut_LinTargetMapper`.

`aut_Mapping` and derived classes

Container classes for mapping algorithm parameters with function to compute mapping.

One `aut_Mapping` derived class is associated to each `aut_MapBuilder` derived class.

`aut_LinearMapping` is similar to `aut_MappingResult`, which is the older implementation. It is used internally in `aut_LinTargetMapper`. Data is then copied into an `aut_MappingResult`.

`aut_GroundTruthCompiler`

Provides functionality to compute animation vectors from ground truth data stored in an `aut_KeyframeData` object. The output is stored in an `aut_LabeldFltMat` object for use in the `aut_MapBuilder` classes.

A description of the process follows. A human anatomy reference model with targets is created in 3D studio Max. Ground truth anchors (spheres named GT_xx) are added to the model to mark the position of tracking nodes. The model is then exported as VRML. The VRML file can be read using the plu_VRMLoader (library PlayerUtils). Target and ground truth data is extracted using the SceneGraph functions and stored into an aut_KeyframeData. Ground truth data exists only for the first target in the VRML file. The function aut_KeyframeData:: setGTDataViaAnchorPos() is used to trace the vertices marked by the ground truth anchors and thus generate ground truth for all targets.

The aut_KeyframeData object is then passed to a properly configured aut_LinTargetMapper to compute the aut_MappingResult, which can be written to a file. The aut_MappingResult created in this way can be used with all models containing targets that model the same expressions, be it human or cartoonish. If decoder data is to be generated for another model, a aut_KeyframeData object has to be prepared from that model. It does not need to contain the ground truth data since this is only used to compute the aut_MappingResult. If the decoder data is generated for the reference model, the original aut_KeyframeData (see above) can be used. It is important that all model parameters that change from neutral to any target are stored in the aut_KeyframeData. This usually includes all vertex positions and the position/angles of the bottom teeth object.

The aut_KeyframeData and the aut_MappingResult are then merged into a att_AnimTagTrafo using the function aut_computeAnimTagTrafo.

The att_AnimTagTrafo contains the decoder configuration parameters that are then exported into the different content files (Pulse/Shout).

The facial animation values or tags may be displacement values relative to neutral face values. Advantageously, 8 to 22 (or more) facial animation values may be used to define and animate the mouth, eyes, eyebrows, nose, and the head angle. Representative facial animation values for the mouth may include vertical mouth position, horizontal mouth position, mouth width, lip distance, and mouth corner position (left and right).

Morphing of a texture map on a deformed three-dimensional head mesh is described in U.S. patent number 6,272,231, titled WAVELET-BASED FACIAL MOTION CAPTURE FOR AVATAR ANIMATION. Imaging systems for acquiring images and image mapping are described in U.S. patent application serial number 09/724,320, titled METHOD AND APPARATUS FOR RELIEF TEXTURE MAP FLIPPING. The entire disclosures of U.S. patent number 6,272,231 and U.S. patent application serial number 09/724,320 are incorporated herein by reference.

Although the foregoing discloses the preferred embodiments of the present invention, it is understood that those skilled in the art may make various changes to the preferred embodiments without departing from the scope of the invention. The invention is defined only the following claims.